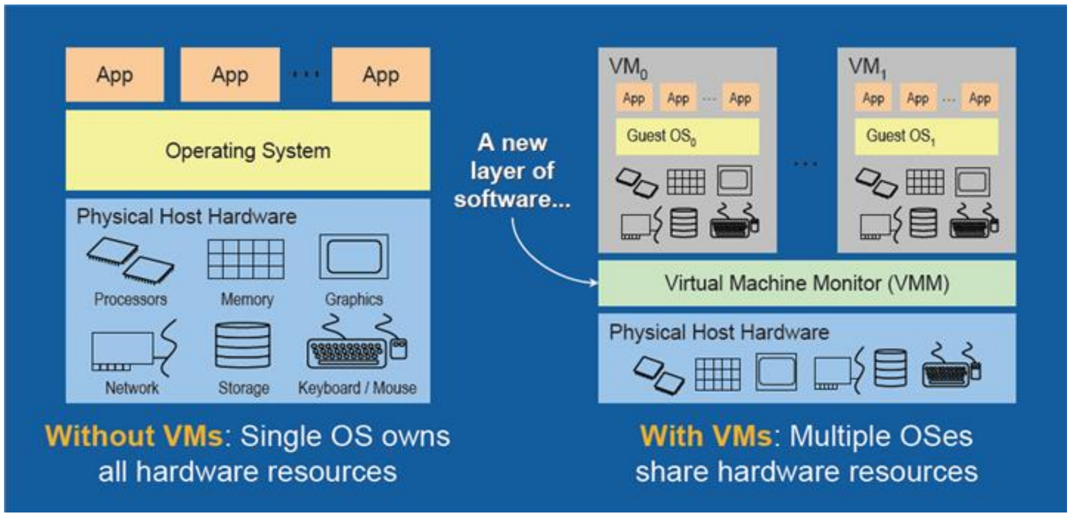
Agenda: Introduction to Containers and Docker

- Understanding VM's and Containers.
- What is Docker?
- Docker Benefits.
- Docker Architecture and Docker Taxonomy.
- The underlying technology

Understanding Virtual Machines and Containers

What is a Virtual Machine

- A virtual machine is a computer file, typically called an image, which behaves like an actual computer – computer within a computer
- It runs in a window, much like any other programme, giving the end user the same experience on a virtual machine as they would have on the host operating system itself.
- Multiple virtual machines can run simultaneously on the same physical computer.
- Hypervisor is the software required for managing VM, emulates the PC or server's CPU, memory, hard disk, network and other hardware resources completely, enabling virtual machines to share the resources.
- The hypervisor (Hyper-v, VMWare, Xen, KVM) can emulate multiple virtual hardware platforms that are isolated from each other, allowing virtual machines to run Linux and Windows Server operating systems on the same underlying physical host.



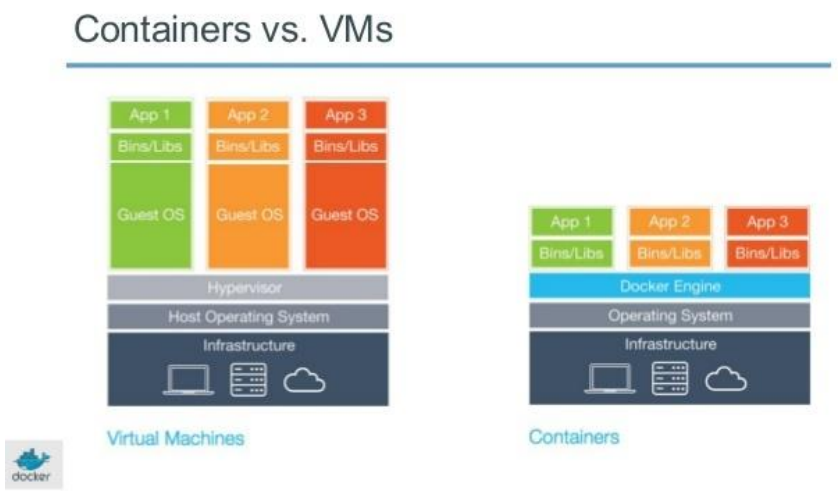
What is a Container?

- Containerization is an **approach** to software development in which **an application or service, its dependencies, and its configuration** are packaged together as a **container image**. You then can test the containerized application **as a unit** and deploy it as a container image instance to the host operating system.

- Placing software into containers makes it possible for developers and IT professionals to deploy those containers **across environments** with little or no modification.
- Containers also isolate applications from one another on a **shared operating system (OS**). Containerized applications run on top of a **container host**, which in turn runs on the OS (Linux or Windows). Thus, containers have a significantly smaller footprint than virtual machine (VM) images.
- Con, agility, scalability, and control across the entire application life cycle workflow. The most important benefit is the isolation provided between Dev and Ops.

Containers vs VMs

Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!



What is Docker and its Benefits

The **Docker platform** uses the **Docker Engine** to quickly build and package apps as **Docker images** created using files written in the **Dockerfile** format that then is deployed and run in a layered **container**.

Benefits of Docker / Container:

1. **Resource Efficiency**: Docker is lightweight and fast. Process level isolation and usage of the container host's kernel is more efficient when compared to virtualizing an entire hardware server using VM.
2. **Fast and consistent delivery of your applications:** By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

3. **Continuous Deployment and Testing**: The ability to have consistent environments and flexibility with patching has made Docker a great choice for teams that want to move from waterfall to the modern DevOps approach to software delivery.

Consider the following example scenario:

a) Your developers write code locally and share their work with their colleagues using Docker containers.
b) They use Docker to push their applications into a test environment and execute automated and manual tests.
c) When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
d) When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Docker is available for implementation across a wide range of platforms:

- **Desktop**: Mac OS, Windows 10.
- **Server**: Various Linux distributions and Windows Server 2019.
- **Cloud**: Amazon Web Services, Google Compute Platform, Microsoft Azure, IBM Cloud, and more.

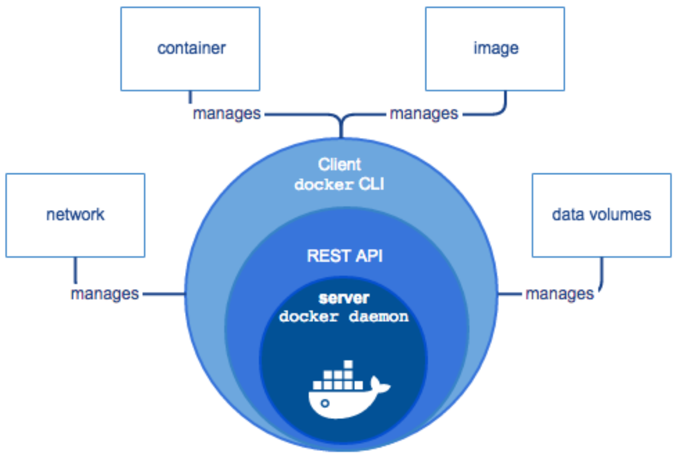
Alternatives to Docker

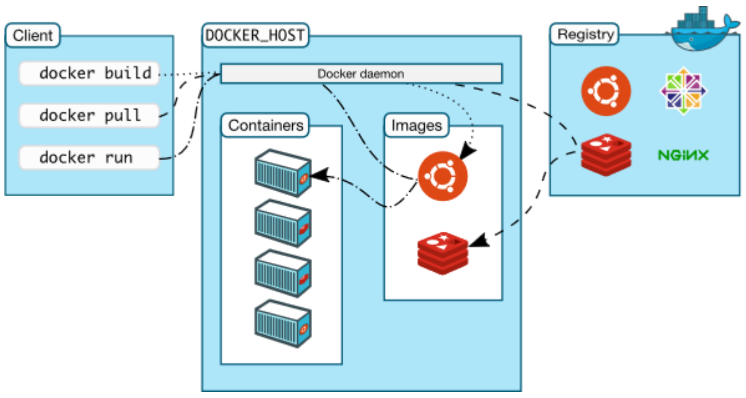
1. Open Container Initiative (OCI)
2. CoreOS and rkt
3. Apache Mesos and Mesosphere
4. Canonical and LXD

Docker Architecture and its Taxonomy

Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a **daemon** process.
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).



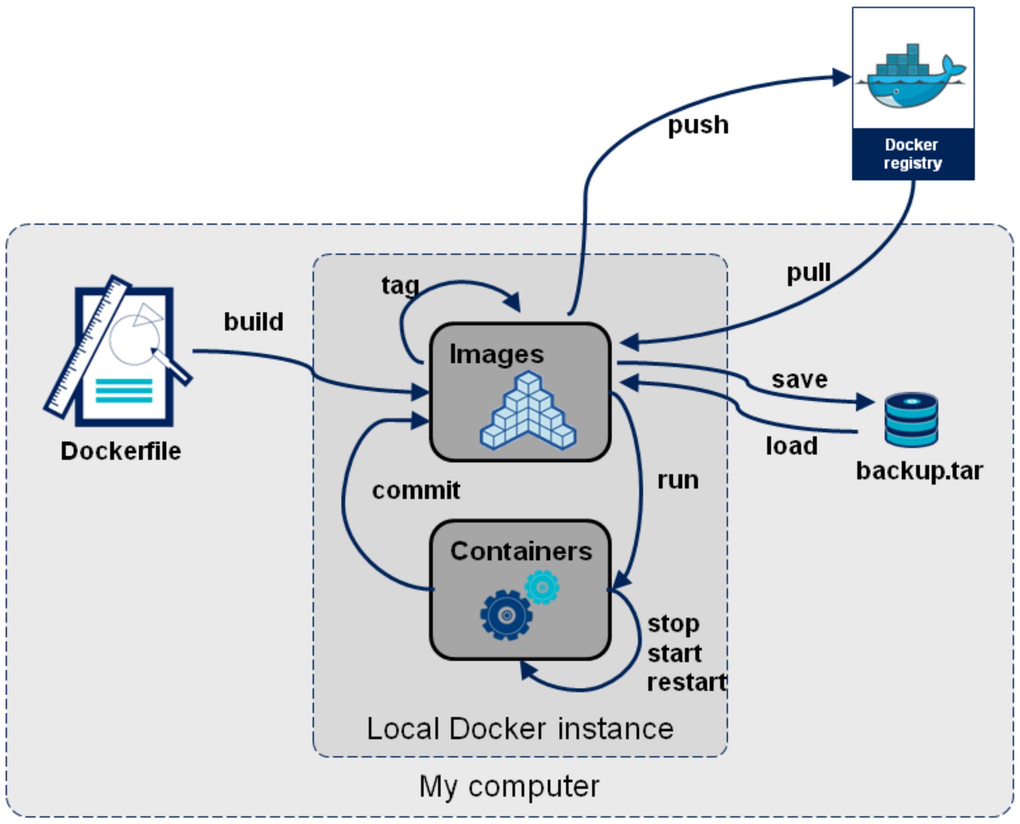


Docker Daemon:

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as **images, containers, networks, and volumes**.
- A daemon can also communicate with other daemons to manage Docker services.

Docker client:

- The Docker client (docker) is the primary way that many Docker users interact with Docker.
- When you use commands such as ***docker run***, the client sends these commands to Docker Daemon, which carries them out.
- The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over sockets or a network interface.
- The Docker client can communicate with more than one daemon.



Docker Image:

- A **package with all of the dependencies** and information needed to create a container. An image includes all of the dependencies (such as frameworks) plus deployment and configuration to be used by a container runtime.
- Usually, an image derives from multiple base images that are **layers stacked** one atop the other to form the container's file system.
- An image is **immutable** after it has been created.
- Docker image containers can **run NATIVELY on Linux and Windows**.
    - Windows images can run only on Windows hosts.
    - Linux images can run only on Linux hosts, meaning a host server or a VM.
    - Developers who work on **Windows** can create images for **either Linux** or **Windows** Images.

Container:

- An instance of an image is called a container. Its an process by itself and each process has its own hostname(IP address) like a Virtual Machine.
- Its an isolated area of the OS with resource usage limits applied.
- The container or instance of a Docker image will contain the following components:

  - An operating system selection (for example, a Linux distribution or Windows)
  - Files added by the developer (for example, app binaries)
  - Configuration (for example, environment settings and dependencies)
  - Instructions for what processes to run by Docker
- A container represents a runtime for **a single application**, process, or service. It consists of the contents of a Docker image, a runtime environment, and a standard set of instructions.
- You can create, start, stop, move, or delete a container using the Docker API or CLI.
- Container makes it possible for each application to have its own networking stack, which means it can have its own IP address, file system and windows registry. In short its an isolated process.

Dockerfile:

- A text file that contains instructions for how to build a Docker image.

Build:

- The action of building a container image based on the information and context provided by its Dockerfile as well as additional files in the folder where the image is built.
- You can build images by using the Docker ***docker build*** command.

Networking:

- Docker implements networking in an application-driven manner and provides various options while maintaining enough abstraction for application developers.
- By default, you get three different networks on the installation of Docker - **none, bridge, and host**. The none and host networks are part of the network stack in Docker. The bridge network automatically creates a gateway and IP subnet and all containers that belong to this network can talk to each other via IP addressing.
- In addition to the built-in default Docker networks, administrators can configure multiple **user-defined networks**. The three types of these networks are **Bridge, Overlay and Macvlan Network.**

Storage Volume:

- Data Volumes provide the ability to create **persistent** storage, with the ability to rename volumes, list volumes, and also list the container that is associated with the volume.
- Data Volumes sit on the host file system, outside the containers copy on write mechanism and are fairly efficient.

Repository (also known as repo):

- A collection of related Docker images labeled with a tag that indicates the image version or OS or any other information.

- Some repositories contain multiple variants of a specific image, such as an image containing SDKs (heavier), an image containing only runtimes (lighter), and so on. Those variants can be marked with tags.
- A single repository can contain platform variants, such as a Linux image and a Windows image.

Tag:

- A mark or label that you can apply to images so that different images or versions of the same image (depending on the version number or the destination environment) can be identified.

Registry:

- A service that provides access to repositories.
- The default registry for most public images is Docker Hub (owned by Docker as an organization).
- A registry usually contains repositories from multiple teams.
- Companies often have private registries to store and manage images that they've created.
- *Azure Container Registry* or AWS Elastic Container Registry are other example.

Docker Trusted Registry (DTR):

- A Docker registry service (from Docker) that you can install on-premises so that it resides within the organization's datacenter and network. It is convenient for private images that should be managed within the enterprise. Docker Trusted Registry is included as part of the Docker Datacenter product. For more information, go to https://docs.docker.com/docker-trusted-registry/overview/.

Compose:

- A command-line tool and **YAML** file format with metadata for defining and running multi-container applications.
- You define a single application based on multiple images with one or more .yml files that can override values depending on the environment.
- After you have created the definitions, you can deploy the entire multi-container application by using a single command **(docker-compose up)** that creates a container per image on the Docker host.

Cluster:

- A collection of Docker hosts exposed as if they were a single virtual Docker host so that the application can scale to multiple instances of the services spread across multiple hosts within the cluster.
- You can create Docker clusters by using Docker Swarm, Mesosphere DC/OS, Kubernetes, and Azure Service Fabric. (If you use Docker Swarm for managing a cluster, you typically refer to the cluster as a *swarm* instead of a cluster.)

Orchestrator:

- A tool that simplifies management of clusters and Docker hosts.
- Using orchestrators, you can manage their images, containers, and hosts through a CLI or a graphical user interface.
- You can manage container networking, configurations, load balancing, service discovery, high availability, Docker host configuration, and more.
- An orchestrator is responsible for running, distributing, scaling, and healing workloads across a collection of nodes.
- Typically, orchestrator products are the same products that provide cluster infrastructure, like Mesosphere DC/OS, **Kubernetes**, Docker Swarm, and Azure Service Fabric.

Docker Community Edition (CE):

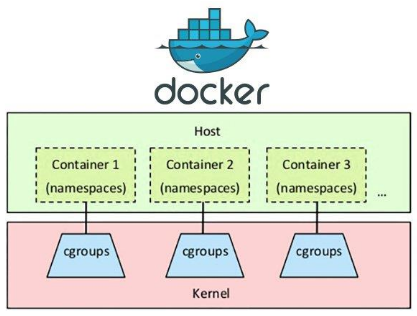
- Development tools for Windows and macOS for building, running, and testing containers locally.
- Docker CE for Windows provides development environments for both Linux and Windows Containers.
- The Linux Docker host on Windows is based on a Hyper-V VM. The host for Windows Containers is directly based on Windows.
- Docker CE for Mac is based on the Apple Hypervisor framework and the xhyve hypervisor, which provides a Linux Docker host VM on Mac OS X.
- Docker CE for Windows and for Mac replaces Docker Toolbox, which was based on Oracle VirtualBox.

Docker Enterprise Edition:

- It is designed for enterprise development and is used by IT teams who build, ship, and run large business-critical applications in production.

The Underlying technology

Docker is written in **Go** programming language and takes advantage of several features of Linux Kernel to deliver its functionality.



Namespaces

Docker uses a technology called namespaces to provide the **isolated workspace** called the container. When you run a container, Docker creates a **set of namespaces** for that container.

Docker uses namespaces of various kinds to provide the isolation that containers need in order to remain portable and refrain from affecting the remainder of the host system.

Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

- The **pid** namespace: Process isolation (PID: Process ID).
- The **net** namespace: Managing network interfaces (NET: Networking).
- The **ipc** namespace: Managing access to IPC resources (IPC: InterProcess Communication).
- The **mnt** namespace: Managing filesystem mount points (MNT: Mount).
- The **uts** namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Control groups (cgroups).

A cgroup **limits** an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce **limits** and **constraints**. For example, you can limit the CPU, Memory, Network I/O or access to filesystem available to a specific container.

To lock a Docker container to the first CPU core: **docker run --cpuset-cpus=0**.

To limit container memory: **docker run –memory=**.

Common control groups

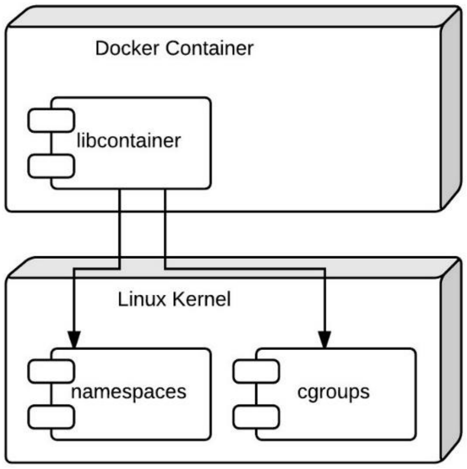
1. CPU
2. Memory
3. Network Bandwidth
4. Disk
5. Priority

Union file systems (UnionFS)

UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers.

Container format

Docker Engine combines the **namespaces, control groups, and UnionFS** into a wrapper called a container format. The default container format is **libcontainer**. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones.


Docker Container
libcontainer
Linux Kernel
namespaces
cgroups